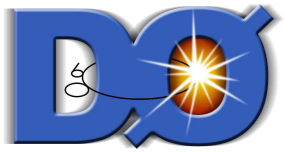
Operations: D0Repro Tools

Design ideas and current features



Daniel Wicke
(Bergische Universität Wuppertal)

Outline

- Introduction
- D0Repro Tools
- Summary

Introduction

DØ Data (and MC) Production naturally falls in to two parts:

1) Performing the execution of defined tasks

SamGrid with help of Runjob and involves:

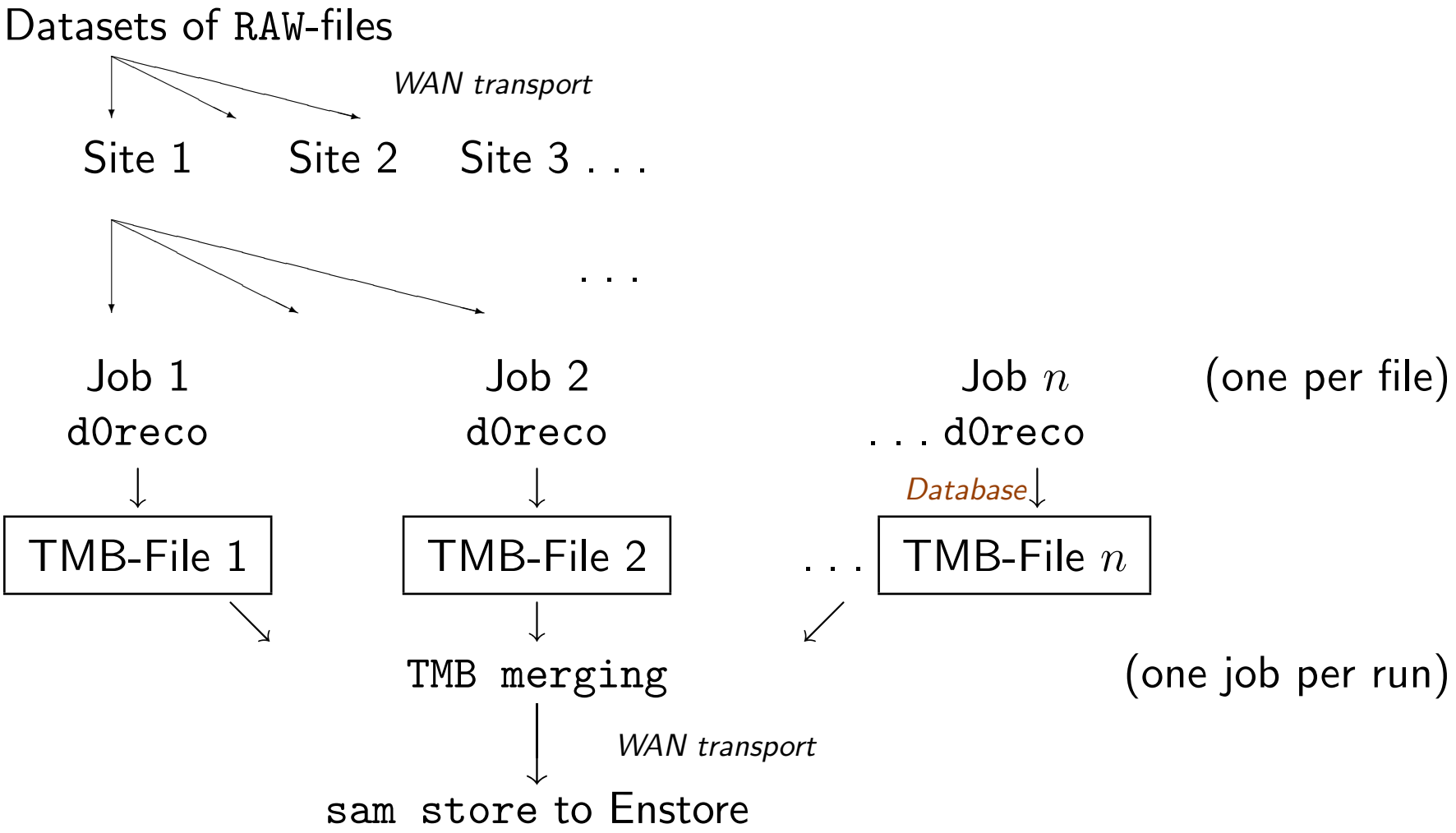
– Shipping jobs to remote CPUs
– Setup of environment for (chain of) executable(s).
– Shipping of input and output files.

2) Defining which tasks need to be performed

Operational Scripts: For data: D0Repro Tools.

– Transformation of production requests into SamGrid JDL.
– Detection of failures
– Automatic creation of recovery jobs.

Application flow (Data reconstruction)



Other applications in data reconstruction chain have similar structure

What can go wrong?

Beside unrecoverable crashes of `d0reco` there will be *random* crashes.

- Network outages
- File delivery failures
- Batch system crashes/hangups
- Worker-node crashes
- Filesystem corruption

To recover we need exact knowledge of what failed and what succeeded.

Book-keeping

1. of succeeded jobs/files
   to assure completion without duplicated events. ⇒ SAM

2. of failed jobs/files
   to trace problems in order fix bugs and to assure efficiency. ⇒ JIM XML

Operational Tools

The application flow starts with a `dataset` to be processed.

Requirements

- determine the full or partial success of processing.
- submit the corresponding (partial) merging jobs.
- determine the full or partial success of merging.
- create and on request submit the recovery jobs for both steps in case of (partial) failure.

To be implemented using

- SAM for obtaining the information about files and
- JIM to submit jobs.

These scripts shall be common to all sites

D0Repro

Basic commands

- Support for certification
- Submission (and recovery) is done by

  `sub_production.py <dataset> <d0release>`

  `sub_merge.py <dataset> <d0release>`
- Determination of production and merge status (poor man's request system)

  `check_production.py <dataset> <d0release>`

  `check_merge.py <dataset> <d0release>`
- Manually modify status of jobs

  `set_status.py [production|merge] [approved|held|finished] <dataset> ...`

Typical workflow:

1)	`sub_production.py ...`	(investigate/retry in case of failures)
2)	`sub_merge.py ...`	(after production is finished; retry if failed)
3)	`set_status.py ... finished ...`	(in case of unrecoverable failures)

Autopilot functionalities

- Investigate status of all active requests `check_all.py`
- Clean completed/finished datasets `clean_completed.py`
- Display status of all active requests and suggests `auto_pilot.py`
  - recover production if less than 5% (50%) failed
  - submit merge if unmerged files exist and last job was production
  - optionally approved additional production jobs (one per automatic merge submission)
- Run commands suggested by autopilot `source Autopilot.sh`

This chain is now run in a loop (with 1 hour delay): `Autopilot.daemon`

Autopilot was built on the experience of reprocessing.
Significantly reduced work-load of operations

More that 90% of the operational work is to chase and fix failures.
Reliable book-keeping is prerequisite to implement these tools.

Summary

- Operational tools define what needs to be run
  - Detection of failures
  - Automatic creation of recovery jobs
  - To avoid duplicate production correct book-keeping is important
- Important design goal is simplicity
  - Operators should need little to no knowledge
  - Few options
  - Further automation is foreseen
- Documentation of current commands available at `http://www-d0.fnal.gov/computing/reprocessing/d0repro/`